Computer Lab 05: Random Variate Generation

In this lab you will write classes implementing three methods for generating *triang(a,b,c)* random variates: Inverse Transform, Composition and Acceptance/Rejection. An instance of one of the classes you write can be retrieved from the `RandomFactory` class and used wherever you have used a `RandomVariate` instance previously (such as your `ArrivalProcess` or `Server` classes).

Concepts

- Inverse Transform method
- Composition method
- Acceptance/Rejection method
- Implementing an interface
- Writing an abstract class

Description

This lab consists of the following parts:

1. Write the abstract base class `TriangleVariateBase`
2. Write the concrete `TriangleITVariate` subclass
3. Write the `TestGenerate` class to test `TriangleITVariate`
4. Modify TestGenerate to create a histogram of generated values
5. Write and test the `TriangleCVariate` subclass of `TriangleVariateBase`
6. Write and test the `TriangleARVariate` subclass of `TriangleVariateBase`
7. Use the classes in the Multiple Server Queue model from Lab 02.

1. Write TriangleVariateBase Class

Define the `TriangleVariateBase` class to implement the `simkit.random.RandomVariate` interface. It should be declared abstract because not all of the methods will be implemented. Since all three versions will use a `RandomNumber` instance for Un(0,1) random variates and will have three parameters, these will be in the base class. Since each version will implement a different algorithm, the `generate()` method will be implemented in the subclasses.

The TriangleVariateBase class has four instance variables that should be define with `protected` access: `double` variables `left`, `right`, and `center`, and `randomNumber`, an instance of `RandomNumber`. Write setters and getters for each of these variables.

The constructor should have zero parameters and should set the randomNumber instance variable, using `RandomNumberFactory.getInstance()`.[1]

Next, write `setParameters(Object[])` and `getParameters()`. The `setParameters()` method should check that the passed-in argument has exactly three elements and that each one is an instance of `Number`. Throw an `IllegalArgumentException` if this is not the case.[2] The `getParameters()` method should simply wrap the three `double` instance variables in an `Object[]` and return it.

---

1. As you did when shuffling the deck of cards earlier in the course.
2. For example, to check for the length of the array:

```
if (params.length != 3){
    throw new IllegalArgumentException("Need 3 elements: " + params.length);
}
```

Finally, you will need to write a public clone() method as follows:

```
public Object clone() {
	try {
		return super.clone();
	} catch (CloneNotSupportedException e) {}
	return null;
}
```

2. Write the TriangleITVariate class

The TriangleITVariate class should extend TriangleVariateBase and should only define three methods: `generate()` and `toString()`. The `generate()` method should implement the Inverse Transform method and use `randomNumber.draw()` to get the Un(0,1) variate.

Generate U ~ Un(0,1)

If $U < (c-a)/(b-a)$

return $a + \sqrt{(c-a)(b-a)\,U}$

else

return $b - \sqrt{(b-c)(b-a)(1-U)}$

The `toString()` method should return a `String` like: `Triangle (1.0, 3.0, 2.0) [Inverse Transform]`. You can put the first part in `TriangleVariateBase` and append the last part in the subclass.

3. Write the TestGenerate Class

Write a pure execution class called TestGenerate that obtains an instance of TriangleITVariate and generates some values from it. Use parameters (1.0, 2.5, 1.5) and seed `CongruentialSeeds.SEED[4]`. The String you need to pass to the RandomVariateFactory is the fully qualified name of the desired class.[1] The first five generated values with this seed should be:[2]

```
Triangle (1.0, 2.5, 1.5) [Inverse Transform]
2.1165808510271344
1.704167240128175
2.3708374474151523
1.739624407394265
1.4006090187354705
```

4. Modify TestGenerate to Create a Histogram

To see the results of your class visually, use the following code:[3]

```
	Dimension screen = Toolkit.getDefaultToolkit().getScreenSize();
	Rectangle window = new Rectangle(400, 300);
	CloseableDataWindow cdw =
		new CloseableDataWindow(rv.toString());
	cdw.setBounds((screen.width - window.width) / 2, (screen.height - window.height)
/ 2,
		window.width, window.height);
	GraphStat gs = new GraphStat("Triangle", 0.0);
```

1. That is, `oa3302.TriangleITVariate`.
2. The first line is from the `toString()` of `TriangleITVariate`.
3. You will have to import `java.awt.*; simkit.stat.*; and simkit.util.*;` The `rv` variable is the `RandomVariate` to be used.

```
cdw.add(gs.initHistogram(true, 1.0, 2.5, 100));
cdw.setVisible(true);
```

In the above code fragment, the first two lines establish the dimensions of the screen and of the data window. The `CloseableDataWindow` class is the shell for displaying the histogram and the setBounds() command places the window at the center of the screen. The `GraphStat` instance produces the histogram, itself with the `initHistogram()` method. The two arguments to `GraphStat`'s constructor are not used in this lab, but are necessary to instantiate a `GraphStat` (the `String` and `double` can be arbitrary, in fact).

The arguments to GraphStat's `initHistogram()` method are as follows:

- `boolean` - `true` if histogram is animated, `false` if not
- `double` - lower limit of histogram
- `double` - upper limit of histogram
- `int` - number of cells in histogram

Finally, to generate the output, write the following loop:

```
for (int i = 0; i < numberToGenerate; i++) {
       gs.sample(0.0, rv.generate());
       cdw.repaint();
}
```

The `sample()` method of `GraphStat` requires a double as its first argument for reasons that do not apply to today's lab. The second argument is the new observation; `GraphStat` will put it in the appropriate bin and update the count. The `repaint()` method will redraw the histogram after the new observation.

5. Write the TriangleCVariate class

The `TriangleCVariate` class extends TriangleVariateBase and uses the composition method to generate a *triang(a, b, c)* random variate. Use this algorithm in the `generate()` method and modify the `toString()` to indicate that it is using the Composition method. Modify TestGenerate to generate a histogram for this method.

6. Acceptance/Rejection Method

Now write a class called `TriangleARVariate` that subclasses `TriangleVariateBase` as generates triangle random variates using the Acceptance/Rejection method with a uniform majorizing function. As before, you will only have to write the `generate()` and the `toString()` methods. Add this method to TestGenerate for a third histogram.

7. Use Triangle Variates in the Queueing Model

Finally use your `TriangleCVariate` and `TriangleARVariate` as interarrival times for the multiple server queue model from Lab 2. The output should look something like this: [1]

```
Multiple Server Queue
       Number Servers: 2
       Service Time Distribution: Triangle (2.0, 5.2, 3.6) [Acceptance/Rejection]
Arrival Process
        Interarrival Times: Triangle (1.0, 3.0, 2.0) [Composition]

Simulation ended at time         2000.0000
There have been 1006 customers arrive to the system
```

1. Use `CongruentialSeeds.SEED[0]` for interarrival times and `CongruentialSeeds.SEED[1]` for service times. The stopping time is 2000.0. You should not have to modify or recompile the `ArrivalProcess` or `Service` classes

```
There have been 1004 customers served
Average Number in Queue          0.2067
Average Utilization              0.9039
```

Output

Histograms that should (hopefully) look roughly like the Triangle pdf. Try different parameters for your Triangle distribution to see their effect. If you change the left and right bounds, change the lower and upper limits in the `initHistogram()` method. For the multiple server queue, output approximately corresponding to the above.

Deliverables

Turn in your source code a picture of your histograms, and the output from the multiple server queue run. To print a picture, select the window with the histogram and press <ALT>-Print Screen. Then open up Wordpad (or Word, if you must) and paste the picture into the document. Finally, print the document.

Frequently Asked Questions

What does it mean to "implement the RandomVariate interface"?

An interface usually has a number of methods that must be defined. For the `RandomVariate` interface, these are the six methods listed on page1. Each method must be written in order to fully implement the `RandomVariate` interface.

I get the following error:

```
oa3302/TriangleVariateBase.java [13:1] clone() in java.lang.Object cannot implement
clone() in simkit.random.RandomVariate; attempting to assign weaker access privileges;
was public
```

You need to add the public `clone()` method to `TriangleVariateBase` as follows:

```
public Object clone() {
       try {
              return super.clone();
       } catch (CloneNotSupportedException e) {}
       return null;
}
```

You do not have to do anything more to the subclasses.

I get the following error:

```
oa3302.TriangleVariateBase should be declared abstract; it does not define generate()
in oa3302.TriangleVariateBase
```

Define `TriangleVariateBase` as abstract (`public` **`abstract`** `class TriangleVariateBase`). It must be declared abstract because the `generate()` method is only implemented in the subclasses.

I get strange errors when I run the histogram part, but the histogram seems to look ok.

If the histogram looks ok, then you can (probably) ignore the errors.

I can't remember the Composition method to generate a triangle variate.

```
Generate U, V ~ Un(0, 1)
if (V < (c - a) / (b - a))
```

```
            Return a + (c - a) √U
else
            Return b - (b - c)√(1 − U)
```

I can't remember the Acceptance/Rejection method for generating a triangle variate.

```
do {
            Generate U, V ~ Un (0,1)
            Y = a + (b - a) V
} while ( (Y < c && U > (Y - a)/(c - a)) || (Y >= c && U > (b - Y)/(b - c) )
return Y;
```